OWASP report

Arjen van der Meer

Introduction

In this report I will be handling every section of the OWASP top 10. This is a list of the 10 most common security faults within applications. I will talk about what each fault means and if and how it is applicable to my application and what I have done or could do to effectively handle these issues.

1. Injection

With injection they mean that when data is send to a server that is going to query a database, a user could instead of giving the expected normal data send a SQL query instead. Such an query would then be executing thus allowing the malicious user to alter or retrieve information of the database where they shouldn't be able to. This is because their send query is executed inside of the query formed by the server to retrieve information from the database. Because I use a database in my application this could be a security risk. However because I use spring boot hibernate this is only possible if I would build native queries. Because I do not do this anywhere in my application my application is safe from SQL injection.

2. Broken authentication

With this they mean that an attacker is able to fake being a different user by for example faking a different users ID. In my application this is not a security risk because I have implemented JWT tokens. These tokens contain the users data and need to be correctly decrypted by the server and checked before a request is passed. That means that without having a correct token for a given user, that can only be formed by having both the users email and password, he can't access any request the user would otherwise be able to.

3. Sensitive data exposure

This means that any sensitive data from a user can be easily accused by others. So for example information about where they live. I want to allow for the user decide which posts which people are able to see. This means that if a user is not supposed to see data of a user that they don't want them to see it is not going to be send to this user. I also ensure that nowhere a user can see another's users email. Ensuring that they can't possibly harass a user by sending them emails. If a user wants to they can always delete their posts, comments and likes. Completely removing them from the database. On top of that a user can decide to remove their account, removing all posts, comments and likes in the process.

4. XML External entities

If a user can upload XML documents and they can insert hostile content in it then they could be able to extract data or execute a remote request from a server as well as make some other attacks. Because I do not use XML anywhere in my application this is not a security risk for me.

5. Broken access control

A broken access control means that someone that is not logged in is able to do certain task where you need to be logged in for. To ensure user don't end up on pages they need to be logged in for the frontend will redirect a user if a user is not logged in. However if a user would still be able to access pages he shouldn't have access too, all request will first be validated by the auth service. Meaning that if they don't have a valid JWT token their requests won't go through. As their JWT token is decrypted an userId and role header is added to the request and send further to the correct service. This service now both knows the userId of the user making the request and their role. Meaning the service can very easily determine if a user should be able to access certain data and can deny them if necessary.

6. Security misconfiguration

This is the most common security issue. It means there are small mistakes in the configuration of the application. This can be things like a misconfigured http header or sending to much information in an error message. The frameworks I use are up to date and I run SonarQube over my project to find possible security risks. If an user tries to login with an email and password, it will only tell them that their credentials are incorrect and not which credentials are incorrect. However, when you try creating an account with a email or username that exists it will return this in an error message. But this only tells you that their exists an account with such an email or username and not to which username or email that account is linked. On top of that you will still need both the password and username to actually log in. An users email is also never shown or send to other users. So only the user itself will know with which email his or hers account is linked.

7. Cross-site scripting XSS

This is where an attacker is able to insert his own code into your website. They could use this for example when a user logs in, sending both the data to the applications backend as to the attacker, giving him the users credentials. By using the latest version of react this vulnerability is very minimal as react escapes XSS automatically, covering most forms of an XSS attack. The data I show is also only pure text. Meaning if an user would add an html tag this would just be printed as a normal piece of text.

8. Insecure deserialization

Insecure desterilization is a more difficult type of attack to perform, however if performed successfully it can be one of the most dangerous forms of attack as it could allow you to run code on the attacking device. My application is likely not at risk as I make sure to validate the users input using the spring framework, thus if an user is sending malicious data it would not be stored.

9. Using components with know vulnerabilities

The fault with using components that have know vulnerabilities is that if an attacker knows you are using these components he can easily exploit the vulnerabilities and attack your application. By using up to date version of the libraries I use it will contain the least amount of possible exploits. To ensure this security risk stays as low as possible it should be important to keep my frameworks as up to date as possible and check for possible deprecated methods or libraries used within my project. As of currently this is not the case.

10. Insufficient logging and monitoring

This means that the application isn't being monitored enough and doesn't keeps track of certain activities that take place. So if there is not much being logged it will be hard to notice if someone is trying malicious stuff. Currently in my application I only log the query's being done and this is mainly for testing purposes. This is a point I could improve my application on as of now, if a user is doing suspicious request these will go by unnoticed. I have however have looked in the past at possible tools that could be used for this, such as intrusion detection tools that keep track of the traffic and try to warn the maintainers of the application if such intrusion is detected.